Amendments to the Claims

1. (Currently amended) A method of implementing a shared message queue using a list structure, comprising the steps of:

defining a list comprising a sequence of list entries, each of said list entries corresponding to a message in said queue and having an associated list entry key, each list entry key corresponding to an uncommitted message falling within an uncommitted key range defining an uncommitted portion of said list and each list entry key corresponding to a committed message falling within a committed key range defining a committed portion of said list; ~~and~~

in response to a request to write a message to said queue, adding a list entry to said list having a list entry key within said uncommitted key range; and

in response to a request to commit said message to said queue, modifying the list entry key associated with said list entry to fall within said committed key range to move said list entry to the committed portion of said list.

2. (Original) The method of claim 1 in which list entries in said uncommitted portion of said list have a defined order as determined by said list entry keys that is preserved when said list entries are moved to the committed portion of said list.

3. (Cancelled)

4. (Currently amended) The method of claim 1 ~~3~~, comprising the further step of:

in response to a request to read a message from said queue, retrieving a list entry whose list entry key has an extreme value in said committed key range.

5. (Original) The method of claim 4 in which said extreme value is a lowest value in said committed key range.

6. (Currently amended) The method of claim 1 ~~3~~ in which said list is a first list, said method comprising the further step of:

in response to a request to read a message from said queue, moving a list entry from the committed portion of said first list to a second list.

7. (Original) The method of claim 6, further comprising the step of:

in response to a request to abort a read of said message from said queue, moving said list entry back from said second list to the committed portion of said first list.

8. (Original) The method of claim 6, further comprising the step of:

in response to a request to commit a read of said message from said queue, removing said list entry from said second list.

9. (Original) The method of claim 1 in which said list entry keys in said uncommitted key range are assigned in order of message priority.

10. (Original) The method of claim 9 in which said list entry keys in said uncommitted key range are assigned in order of arrival time for messages of a given priority.

11. (Original) The method of claim 1 in which each list entry key has a more significant portion indicating the list portion to which the corresponding list entry belongs and a less significant portion indicating the order of said list entry in said list portion.

12. (Original) The method of claim 1, further comprising the step of:

in response to a request from a requester to wait for a message in said queue:

detecting a change in state of the committed portion of said queue from an empty state to a not-empty state;

in response to detecting a change in state of the committed portion of said queue from an empty state to a not-empty state, notifying said requester of said change of state.

13. (Currently amended) Apparatus for implementing a shared message queue using a list structure, comprising:

means for defining a list comprising a sequence of list entries, each of said list entries corresponding to a message in said queue and having an associated list entry key, each list entry key corresponding to an uncommitted message falling within an uncommitted key range defining an uncommitted portion of said list and each list entry key corresponding to a committed message falling within a committed key range defining a committed portion of said list; ~~and~~

means responsive to a request to write a message to said queue for adding a list entry to said list having a list entry key within said uncommitted key range; and

means responsive to a request to commit said message to said queue for modifying the list entry key associated with said list entry to fall within said committed key range to move said list entry to the committed portion of said list.

14. (Cancelled)

15. (Currently amended) The apparatus of claim 13~~-14~~, further comprising:

means responsive to a request to read a message from said queue for retrieving a list entry whose list entry key has an extreme value in said committed key range.

16. (Currently amended) The apparatus of claim 13~~-14~~ in which said list is a first list, said apparatus further comprising:

means responsive to a request to read a message from said queue for moving a list entry from the committed portion of said first list to a second list defined for said queue.

17. (Original) The apparatus of claim 16, further comprising:

means responsive to a request to abort a read of said message from said queue for moving said list entry back from said second list to the committed portion of said first list.

18. (Original) The apparatus of claim 16, further comprising:

means responsive to a request to commit a read of said message from said queue for removing said list entry from said second list.

19. (Original) The apparatus of claim 13, further comprising:

means responsive to a request from a requester to wait for a message in said queue for detecting a change in state of the committed portion of said queue from an empty state to a not-empty state and, in response to detecting said change in state, notifying said requester of said change of state.

20. (Currently amended) A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for of implementing a shared message queue using a list structure, said method steps comprising:

defining a list comprising a sequence of list entries, each of said list entries corresponding to a message in said queue and having an associated list entry key, each list entry key corresponding to an uncommitted message falling within an uncommitted key range defining an uncommitted portion of said list and each list entry key corresponding to a committed message falling within a committed key range defining a committed portion of said list; ~~and~~

in response to a request to write a message to said queue, adding a list entry to said list having a list entry key within said uncommitted key range; and

in response to a request to commit said message to said queue, modifying the list entry key associated with said list entry to fall within said committed key range to move said list entry to the committed portion of said list.

21. (Cancelled)

22. (Currently amended) The program storage device of claim 20 ~~21~~, comprising the further step of:

in response to a request to read a message from said queue, retrieving a list entry whose list entry key has an extreme value in said committed key range.

23. (Currently amended) The program storage device of claim 20 ~~21~~ in which said list is a first list, said method comprising the further step of:

in response to a request to read a message from said queue, moving a list entry from the committed portion of said first list to a second list defined for said queue.

24. (Original) The program storage device of claim 23, further comprising the step of:

in response to a request to abort a read of said message from said queue, moving said list entry back from said second list to the committed portion of said first list.

25. (Original) The program storage device of claim 23, further comprising the step of:

in response to a request to commit a read of said message from said queue, removing said list entry from said second list.

26. (Original) The program storage device of claim 20, further comprising the step of:

in response to a request from a requester to wait for a message in said queue:

detecting a change in state of the committed portion of said queue from an empty state to a not-empty state;

in response to detecting a change in state of the committed portion of said queue from an empty state to a not-empty state, notifying said requester of said change of state.